CompliancePal: A Tool for Supporting Practical Agile and Regulatory-Compliant Development of Medical Software

Stirbu, Vlad

IEEE
2020-05

Stirbu , V & Mikkonen , T 2020 , CompliancePal: A Tool for Supporting Practical Agile and Regulatory-Compliant Development of Medical Software . in 2020 IEEE International Conference on Software Architecture Companion . IEEE , pp. 151-158 , IEEE International Conference on Software Architecture Companion , Salvador , Brazil , 16/03/2020 . https://doi.org/10.1109/ICSA-C50

http://hdl.handle.net/10138/322904
https://doi.org/10.1109/ICSA-C50368.2020.00035

unspecified

CompliancePal: A Tool for Supporting Practical Agile and Regulatory-Compliant Development of Medical Software

Vlad Stirbu
CompliancePal
Tampere, Finland
vlad.stirbu@compliancepal.eu

Tommi Mikkonen
University of Helsinki
Helsinki, Finland
tommi.mikkonen@helsinki.fi

***Abstract*—As digital transformation affects more and more industries, the increased role of software and the skills required to develop software trigger a ripple effect. Entire industries, where regulations and government standards play an important role (e.g. health care, avionics, etc.), have used long development cycles that relied on detailed up-front planning before advancing to any detailed decision. In contrast to this mindset, agile software development has proven to deliver results that satisfy customers needs faster than traditional waterfall methodologies. The lack of detailed upfront planning and fast delivery cycles have led to situations where the use of agile became synonymous with lack of documentation and poor quality, and hence the perception that the approach is not suitable for regulated systems. In this experience paper we describe the implementation of a service that integrates medical device software compliance specific activities such as architectural design and limited risk management into the daily agile practices of a software development team.**

I. Introduction

Digital transformation is a phenomenon where digital technology gets integrated in all areas of a business and the society. An obvious result of the transformation is that a company engaged in this path becomes a software business, which fundamentally changes how the company operates and delivers value to its customers. The transformation is typically accompanied by a cultural change that requires the organization to challenge status quo through continuous experimentation and getting familiar with failure.

Digital transformation poses a significant challenge for businesses operating in heavy regulated environments such as health care. The agility enabled by modern software development methodologies, open source software, and the increasing use of cloud infrastructure capabilities is at an apparent conflict with the practices that are used to implement the regulatory frameworks, which have been optimized for the old world that relied on heavy up-front planning, followed by rigorous execution of these plans.

This paper presents CompliancePal, a service that provides templates and workflows that enables an agile team to perform, in an automatic fashion, a subset of the compliance activities required by the regulatory framework specific for medical device software. Although the idea of using modern automation practices, commonly known as *DevOps*, has been explored in academia [1], we approach the problem domain from a practical software engineering perspective in an industrial setting. The paper is a continuation of our previous work [2], where the tool idea was presented but only at a principal level.

The paper is structured as follows. Section II introduces modern software developments and practices that enable high velocity delivery of features. Section III provides an overview of regulatory landscape relevant for medical device software development. Section V describes the service implementation, followed by a discussion of the results in Section VI. Concluding remarks are presented in Section VII.

II. Modern Software Development Practices

Agile software development [3] is a lightweight approach for developing software, where the requirements and the solution evolve through collaboration between the development team and the beneficiary of the solution. The development team is typically cross-functional and self-organizing. The development team starts with an initial plan and design that evolves through a rapid cycle of releases and continuous improvement. The approach promotes flexible response to change over strictly following plans, and it has been sometimes misinterpreted as a series of ad-hoc decisions rather than a disciplined engineering methodology. In this paper, we build on the disciplined interpretation of agile software development and overlook the different deviations such as ScrumBut [4] and other (mis-)practices that are often adopted.

Scrum [5] is one of the most commonly used agile framework for software development. The methodology defines two special roles within the development team: the *product owner* that represents the voice of the stakeholders and customers, and the *scrum master* that facilitates the scrum and is responsible with removing impediments that can hamper the ability of the team to deliver on their goals. The development team works in small time-boxed increments called *sprints* and takes work items from an ordered list of product requirements called *product backlog*. Each sprint is bounded by a *planning* and a *review* session. During the sprint, the team holds *daily scrum* sessions to evaluate progress and identify impediments.

DevOps [6] combines practices and tools with cultural philosophies that increases the ability of an organization to deliver applications at high velocity. The DevOps practices and culture are aligned with and complement agile software development practices by integrating, testing and deploying applications at a rapid pace. For example, monitoring in real-time the behavior of the applications in production and acting if not performing within the desired quality parameters, creates a fast feedback loop.

The capabilities of modern infrastructures exposed via application programming interfaces (APIs) are leveraged into a set of tools that allow a high level of automation throughout the life-cycle of the application. The use of software development practices for handling the infrastructure, such as version control, allows the changes to be handled in a standard and controlled way, resulting in repeatable and consistent deployments that can be easily rolled back.

Agile software development and DevOps create an environment in which *experimentation* can thrive [7]. By having the mundane tasks of testing, integrating, packaging and deploying automated, the teams can focus their attention on bringing new features to the customers faster. Anyone in the team can perform these operations as needed without required additional expertise.

III. Medical Device Software Regulations

The regulations covering medical device software fall into two broad categories: information handling and safety. The regulations related to information handling are Health Insurance Portability and Accountability Act (HIPAA) [10] in Unites States and General Data Protection Regulation (GDPR) [8] in the European Union. HIPAA is a medical sector regulation that defines what constitutes *protected health information*, its use and disclosure when several health providers are involved in the care process. GDPR is a generic data privacy framework that defines how personal information is collected and used. To comply with HIPAA and GDPR regulatory frameworks, a service provider that implements part of the functionality using software must establish procedures for handling the relevant information. These procedures typically get materialized into technical requirements, which have to be implemented in software, or standard operating procedures, which have to be followed by the staff that interacts with the protected information. The regulations extend to business associates that process or handle protected information, which have to comply themselves.

The safety of medical devices or services is regulated in Unites States by the Food and Drug Administration (FDA) and Medical Device Regulation (MDR) in European Union. International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) have developed harmonized standards that contain guidance on the processes and requirements that must be followed when developing software so that relevant regulatory authorities accept medical products in the respective markets.

For example, ISO 13485 [12] specifies the requirements for a quality management system that allows an organization to demonstrate its ability to provide medical devices and related services that consistently meet customer and applicable regulatory requirements. Further, ISO 14971 [13] specifies the processes that a manufacturer must follow to identify the hazards associated with medical devices, to estimate and evaluate the associated risks, to control these risks, and to monitor the effectiveness of these controls. Lastly, the life cycle requirements that must be followed by an organization where software is embedded or is an integral part of the final medical device are defined in IEC 62304 [11]. The requirements are envisioned as a set of processes, activities and tasks that establish a common framework for medical device software life cycle processes.

IV. Medical Device Software Agile Challenges

Although the processes, activities, and tasks defined in the IEC 62304 are specific in what they have to accomplish, the standard is not opinionated on how they are actually implemented. This situation leaves a high degree of freedom for affected organizations to select the software development methodologies best suited for their needs, such as waterfall, agile, or hybrid. However, the implementation is relatively straightforward for traditional waterfall software development, as the upfront detailed planning can incorporate all the steps required. For practical cases, this has often been misinterpreted as a requirement regarding the development approach, although that is not the intention.

Showing compliance to the standard gets complicated when an implementation relies on agile and DevOps methodologies. There, the objective is on delivering customer features at high velocity while maintaining the organizational ability to react to changes at reduced costs. As the plans are detailed and revised regularly based on the learnings gathered during the implementation and releases to the customers, there is an increased possibility for rifts to appear between what is documented as part of the compliance activities and the actual realities of the software implementation.

A. Definition of Done vs. Agile Manifesto

The *definition of done*[1] is a list of criteria, agreed by the agile team, that must be met before a product increment is considered complete. Often, the documentation is formally included in this list.

However, if the team has a low awareness of the constraints of the regulated domain and the relevant regulatory legislation and standards – or, as often happens, is influenced by a too strict interpretation of the agile manifesto clause of *working software over comprehensive documentation* – a situation may emerge, where the created documentation lacks critical information required to create regulatory documentation. However, even if the domain experience cannot be easily compensated, the shortcomings can be mitigated by having

[1] https://www.agilealliance.org/glossary/definition-of-done

clear guidelines and scheduling explicitly time for producing the documentation, instead of hoping the developers are doing it because they are *professionals*.

B. Documentation Sprint

The *documentation sprint* (depicted in Figure. 1), is a relatively common practice in agile teams developing medical device software. In addition to standard Scrum, there is a special sprint allocated to creating the documentation required by regulations before a major product release or a milestone. To meet this objective, the team is extended with specialist compliance officers that have deep knowledge of relevant regulatory legislation and standards. As a result, the team's daily activities are dominated by the compliance officers that collect the information required to create the necessary reports.

The compliance officers organize review meetings or conduct interviews with the team members involved in software development, if the documentation produced during the regular sprints is not satisfactory. The accuracy of the collected information during this phase depends on factors varying from the availability of the original persons that designed or implemented a particular software items or simply how well they remember past decisions. This routine is repeated till the required documentation is completed, which affects the team cadence. To make matter worst, the compliance officers are staffed as contractors only during the duration of this sprint, which further increases the friction within the team.

V. CompliancePal: Medical Device Software Development Goes Agile

When developing medical device software, the agile team activities and practices must be aligned with the software lifecycle processes defined in IEC 62304.

A. Service Integration Model

The environment, depicted in Figure 2, mirrors the daily routine of an agile development team practicing the scrum methodology. Among the agile team we emphasize three relevant roles: the software developer, the architect and the compliance officer. The code produced by the team is managed using a Git repository hosted on GitHub[2]. The compliance checks are performed by our service that extends the standard GitHub workflows using the *Apps*[3] integration methodology. Possible compliance problems are brought to the attention of the team via dedicated chat room hosted in Slack[4].

The architect, which can also be a one of the software developers, documents the software components that implement the product requirements, defines their hierarchy, and how they interact with each other using a lean software architecture model. The resulting documentation is managed in a repository in GitHub, in a similar fashion as the rest of the code produced by the team.

The software developer workflow consists of developing new functionality, test it locally, then commit changes to the local git repository. Following the team practices to merge completed new functionality to the common code base, the developer pushes the changes to the remote GitHub shared repository. GitHub notifies its service integrations upon receiving new changes. Our service receives the change notification and performs code analysis to identify new SOUP components in the new change set. The service updates the progress of the check using the GitHub Checks[5], an API that allows third parties to report rich statuses, or annotate lines of code with detailed information, rather than a mere binary pass/fail status. Further information can be obtained by following a custom *web link* to our service. The developer is able to follow the progress as well as the rich status using the familiar GitHub user interface.

The compliance officers are able to react to the detected problems that require their attention via the team communication channel. Following the link, the officer can handle the problem in the service UI. This way, the compliance officer does not have to be familiar with how to use the GitHub service.

B. Lean Software Architecture

One of the key premises of agile development is that working software is more important than complete documentation. During the transition to agile ways of working, documentation and especially documentation done using Unified Modeling Language (UML) tools were the first victims when observed from grass-root level. The high level of expertise required to use the modeling tools, the level of details, and the effort to maintain the models in sync with the implementation was perceived as an impediment. In a relatively short interval, UML was abandoned in favour of adding UML-like diagrams into wikis. The trade-off, which was intended to increase the team speed, lead often to a similar level of obsolete documentation. More, as these diagrams are produced from wiki friendly domain specific languages like PlantUML[6], or by visual drawing tools like Gliffy[7], they lack the affordances offered by the models created by the UML tooling. Documenting software architecture using diagrams, which cannot be used programatically downstream, further isolates the architecture work from the rest of the team. In a DevOps dominated world, the manually maintained wikis have not chance other than to fail behind and rot.

The approach followed by CompliancePal is very practical in nature. We aim to achieve two objectives: provide value to the team from allocating resources for architectural work, and establish a baseline for performing the compliance activities and tasks required by IEC 62304.

We decided to adopt the C4 model [14] as the starting point of our solution rather than creating a bespoke solution. The model has a track record of being used successfully

[2]https://github.com
[3]https://developer.github.com/apps/
[4]https://slack.com
[5]https://developer.github.com/v3/checks/
[6]https://plantuml.com
[7]https://www.gliffy.com

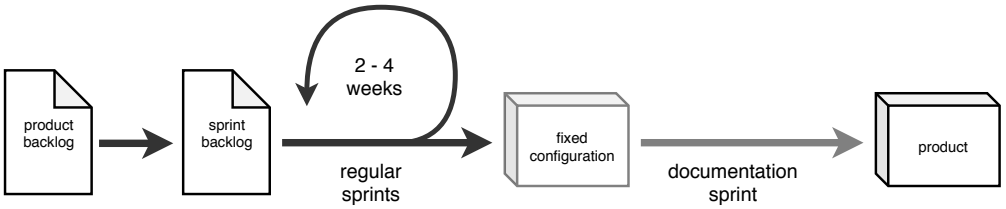


Figure 1. Documentation sprint

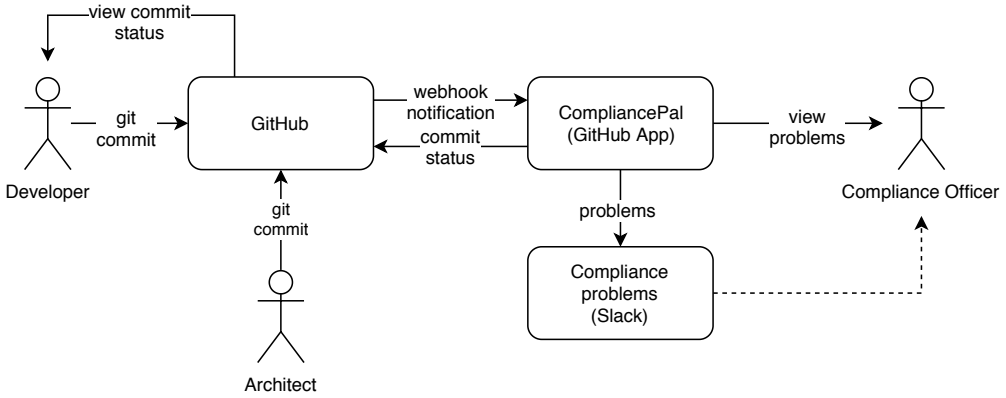


Figure 2. Service model and compliance workflows

by agile teams, and has a growing ecosystem of tools build around it. It can be used for the upfront design of a software system and for retrospective documentation, knowledge sharing and learning of how an existing software system works, making it both an effective design tool within the team, and communication tool outside the team. Its hierarchical set of software diagrams for context, containers, components and code provides different levels of abstraction, each targeted at different audiences. The *context* diagram emphasizes the software system and its *interactions* with the outer world, those being users or other software systems. The *container* diagram describes how the software is *packaged* into runnable artifacts. The *component* diagrams highlights the parts used to assemble individual runnable artifacts. Finally, the *code* diagram shows a component's implementation details.

An architecture described using the C4 model complies with a simple meta-model containing elements and relationships. Besides the four software elements, the meta-model includes a *person* element that corresponds to users. The relationships capture interactions between the elements.

The C4 model is well aligned and supports the software architecture activities specified by IEC 62304. The standard requires that the device manufacturer decomposes the software into systems, items, and units. Instead of creating an entirely new decomposition, we augmented the C4 meta-model[8] with a corresponding *decomposition* property, with possible values described in Table I. The service renders the original C4 diagrams and the specific ones required for producing IEC 62304 documentation from the same architecture model. An example,

Table I
C4 META-MODEL EXTENSIONS

C4 element type	*decomposition*	Observations
Software System	System	
Container	Item	
	Unit	Not decomposed further if SOUP
Component	Item	
	Unit	Not decomposed further if SOUP
Code	Unit	

illustrated in Figure 3, is the software decomposition diagram that captures the hierarchical structure of the CompliancePal as a software system.

The software architecture model is serialized and stored in a git repository hosted in GitHub with the rest of the software assets. The arrangement allows the architecture to be versioned using normal git facilities, using the same life cycle procedures as for any code. The user is able to update the software architecture by pushing new versions via git or by editing model with the browser based user interface.

C. Continuous SOUP Analysis

The C4 hierarchical model of the software architecture conveys, via the *location* property of an element, information about the components developed by the team (e.g. internal) and the components developed by a 3rd party (e.g. external). The second category, known as software of unknown provenance[9] (SOUP), is a major source of concern as it was not developed

[8]https://c4model.com/#metamodel

[9]https://en.wikipedia.org/wiki/Software_of_unknown_pedigree

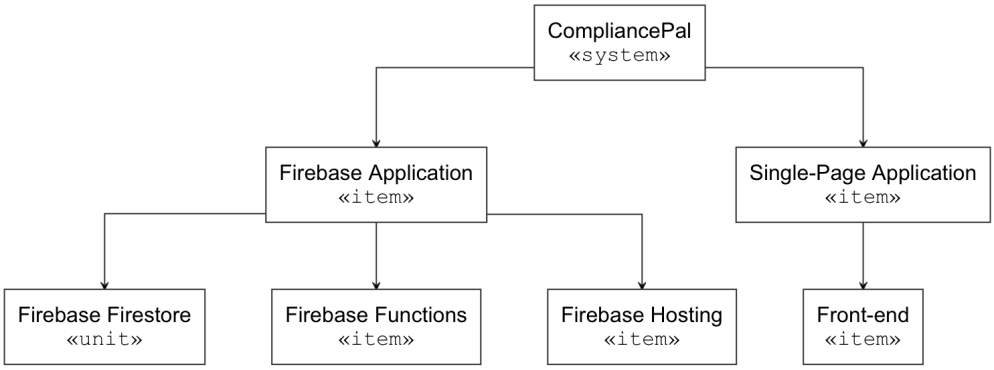


Figure 3. Software decomposition diagram according to IEC 62304

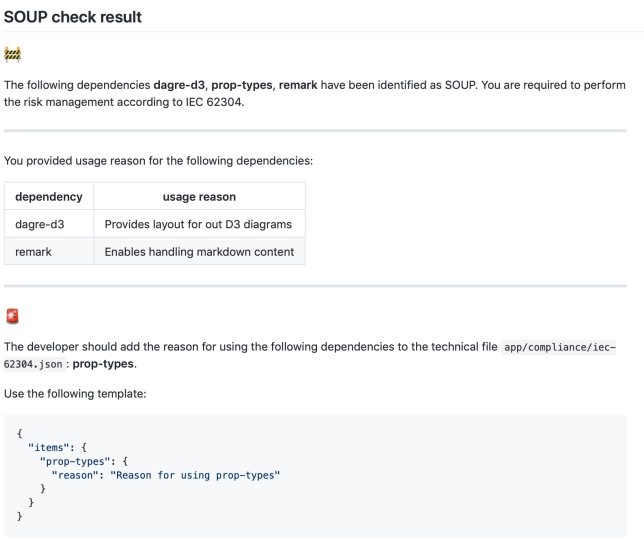


Figure 4. Developer view of SOUP check in GitHub UI

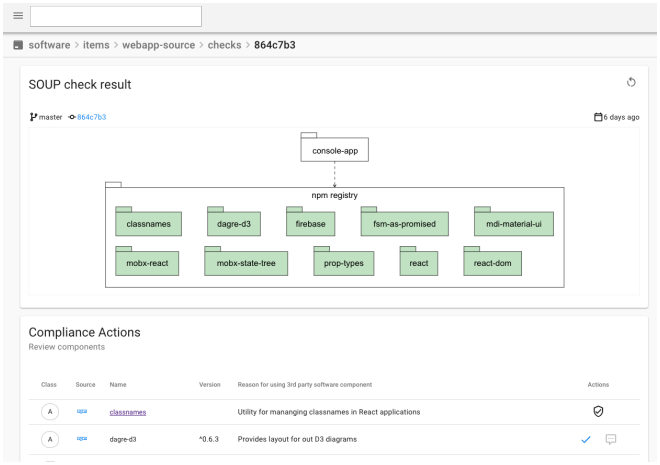


Figure 5. Compliance officer view of SOUP check in CompliancePal UI

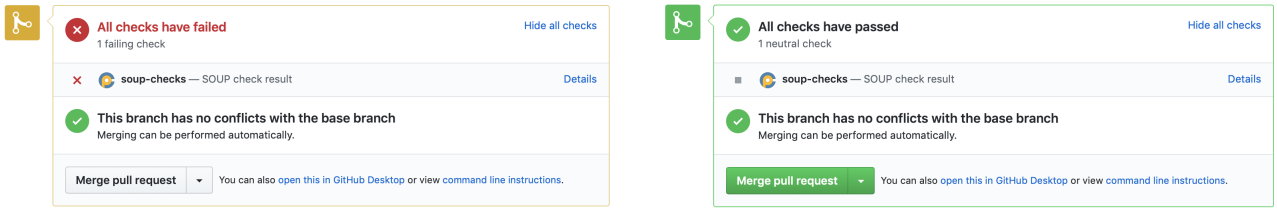
following the medical device software life cycle required by IEC 62304. A manufacturer using SOUP components must perform specific risk management activities that ensure that faults in respective components do not have an adverse effects. Automatically tracking the SOUP components ensures that all such components are identified and handled accordingly, reducing the cognitive load on the software developers and compliance officers.

CompliancePal performs automatic SOUP management for JavaScript assets, such as NodeJS[10] applications or browser applications developed with React[11] or Angular[12]. Once a code repository is linked with a software item defined in the software architecture, the CompliancePal tool listens to code updates delivered by GitHub via push events. For each such event, the service fetches the repository tarball and performs static code analysis to identify changes in SOUP components used. The result of the analysis contains three sections: the full list of SOUP components used, the subset that has been handled and the subset that requires additional information. The SOUP check report is posted to the corresponding commit using the Checks API and can be inspected by the developers using the GitHub user interface (see Figure 4).

If more information is needed for specific packages, the developer follows the instructions in the check report and provides at least the reason for using the specific SOUP component. The update is delivered as a new commit to GitHub, which notifies CompliancePal to re-analyse the code. When the new functionality is ready to be merged in the common base, the developer opens a pull request.

The compliance officer investigates the SOUP changes using the CompliancePal user interface and checks that the information provided by the developer is satisfactory (see Figure 5). If more information is needed, the compliance officer can open a review request indicating what is missing. When the compliance officer considers that the information related to a SOUP component is detailed enough, he can approve the SOUP component. Once a SOUP is approved, the component is added by the service to the architecture model and appears in the SOUP list without further action from the user. When

[10]https://nodejs.org/en/
[11]https://reactjs.org
[12]https://angular.io



(a) check failed

(b) check success

Figure 6. SOUP compliance quality control via GitHub pull request

all SOUP components in the pull request are approved the check status changes to success. At this point, the branch can be merged into the baseline using the GitHub user interface (see Figure. 6).

D. Development Workflow Models with Compliance Quality Control

Git was originally designed for coordinating work among programmers. It provides distributed version-control for tracking changes in source code during software development, but it can be used to track changes in any set of files. Being distributed by design, each Git directory is a full-fledged repository with full history and version tracking abilities, without requiring network access or a central server. The default Git functionality has rich affordances that allow teams to build workflows that enables multiple developers to work efficiently on a single common base.

The workflows are typically build upon *pull requests* [15], a feature pioneered by GitHub. Opening the pull request signals that a developer is ready the merge the changes that have made into their local repository into the main project. The process includes a *review* phase in which the reviewers can comment with improvement ideas as seen necessary. The creator responds with new changes till issues raised in the comments are addressed. The review discussion is followed by *merging* the changes into the common code base.

This ability allows teams to use Git in multiple development workflow models, the most popular being *Git flow* and *trunk-based development*. In Git flow, there is one *development* branch with strict access. Developers create feature branches from this branch and, once their work on the feature is done, they open pull requests. Following the review discussion, the agreed changes are merged. When enough the features scheduled for the next release are done, a release candidate branch is created and after testing and bug fixes are applied it is merged into the *master*. In the meantime, the new features are developed in the development branch. In the trunk-based development model, all developers collaborate on a single branch, typically the *master*. The development is done in short-lived feature branches which are merged into the trunk following the review step.

Both development models have merits and are appropriate depending various aspects such as the homogeneity of the team, or the development stage of the product. From the compliance with medical device software regulation perspective, we are interested in the what software artifacts are created and not the development methodology used to create them. The CompliancePal quality control accommodates both development workflows, enforcing the SOUP checks in the integration branch, *development* and *master* respectively. The selection of the integration branch is done during the configuration of the software item' source repository.

E. Software Assets Organization Strategies

Git allows team to organize the content of the repository in many different ways. At one end of the spectrum we have the *multi-repo* strategy in which every project is maintained into a separate repository. At the other end of the spectrum we have the *monorepo* where a large number of projects are maintained within a single repository. Each strategy comes with its own advantages and disadvantages and is up to the teams to select which strategy is the most appropriate. Our service is interested in performing compliance checks regardless of the strategy employed by the manufacturer. The selection of the strategy is performed during the configuration of the software item' source repository. The procedure allows CompliancePal to recognize if the change sets delivered in a push event are relevant or not.

VI. Discussion

In this section we discuss the experimental result from two perspectives. in the context of the full software development lifecycle described by IEC 62304. Then, we explore the most practical way to incorporate the existing rich ecosystem of tools used widely by agile and DevOps practicing teams.

A. Software Architecture and SOUP Checks as Enablers for Risk Management

IEC 62304 defines three threat classes that convey information about the level of harm that the recipient of the medical device that includes software can experience if used according to the intended use specified by the manufacturer: A does not result in any injuries, B can lead to non-serious injury, and C can lead to serious injury or death.

To evaluate in which category the software fits, the product is decomposed into *software systems* that consist of integrated collections of *software items*. The items can be decomposed further into *software units*. The manufacturer decides on granularity of items and units. The risk classification is by default inherited from the parent component and can be changed by

performing risk analysis at component level. If following the risk analysis a child component is classified with a higher risk than its parent, the classification of the parent components are elevated recursively up to the software system.

The augmented C4 model described earlier serves as the basis to perform the risk analysis. Capturing the software architecture and its evolution as it happens, corroborated with the information about external risks introduced by SOUP components, enables the compliance officers to conduct the risk analysis and classification without further assistance from the development team. These activities can be performed as they happen, if compliance officers are available during the development phase. For smaller teams, where compliance officers join the team only before major milestones, the compliance activities can be performed in retrospective by time traversing these change sets.

To capture the result of the risk analysis, we have expanded further the C4 meta-model with the *class* property. With the decomposition and the classification we can automate the enforcement of the classification constraints required by IEC 62304. The work of the compliance officers is eased as the tool points out which software components classification has to be reassessed, eliminating a potential source of human error.

B. Traceability and Verification in an Open Ecosystem

Traceability refers to the ability of tracking requirements implementation as they are decomposed into product requirements, system requirements, and finally into software requirements. Verification refers to the ability to verify the actual implementation state of the mentioned requirements.

Modern agile teams have a plethora of tools to choose from when implementing traceability and verification. For example, most web based Git providers offer a basic level of requirements management via the bundled *issue* trackers. The issues can be referenced using short links, such as `user/repository#12` where the `user` represents the username or the organization, `repository` represents the repository where the software assets is developed, and `12` represent the issue number. Teams that are not satisfied with the bundled issue trackers use dedicated solutions, such as JIRA[13], that have the ability to track issues with branches. These smart links convey information that can be used for traceability in the IEC 62304 context.

The teams practicing DevOps can employ internal (e.g. Jenkins[14]) or external services (e.g. TravisCI[15] or CircleCI[16]) that are typically used to run tests before code can be merged into the common base, package software artifacts that are ready to be released, or ready to be delivered to production environments. These test suites can be mapped to the *unit*, *integration* or *system* tests mentioned in IEC 62304.

Issue trackers and continuous integration pipelines are used in various combinations to automate team unique workflows and operating environments. They have been selected primarily for those purposes, not with the explicit goal of producing the documentation required by IEC 62304. Each tool's maturity and rich functionality makes it unlikely that it will be replaced by a another dedicated tool that handles only traceability and verification. A more sensible approach is to leverage their API integration capabilities and access the necessary information that is already available in the tools' internal databases. This way the information is transformed into a form suitable for producing traceability and verification reports. However, due to the heterogeneity of existing and emerging issue trackers and continuous integration it would be difficult to develop a tools that can fully handle all combinations.

C. IEC 62304 Clauses Coverage Considerations

Manufacturers that develop medical software have to comply with specific clauses defined in IEC 62304 depending on the intended use of their product (see Figure 7a). Generally, design documentation and testing is not required for products that fit within Class A. Products covered by class B require software design documentation and testing, while the ones in class C require thorough design documentation and testing, including the interfaces with hardware components.

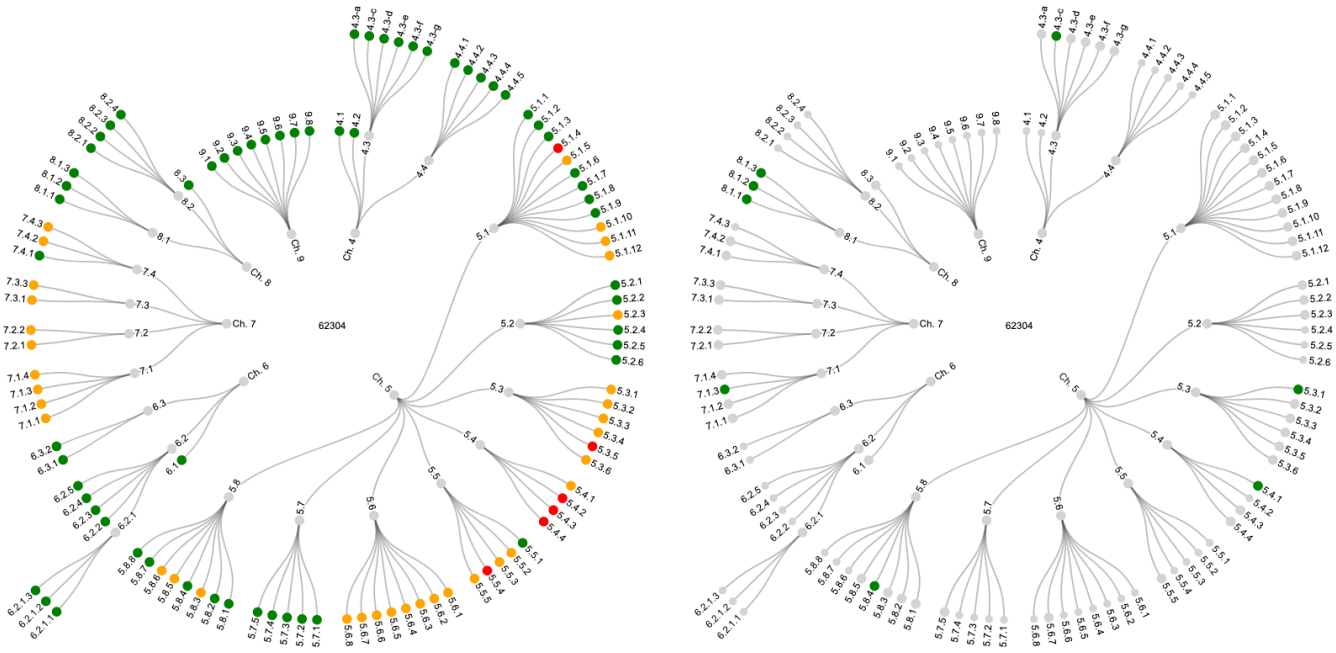
IEC 62304 covers the entire software development lifecycle of a product. In this context, our service functionality covers a rather limited set out of the entire collection defined in IEC 62304. The initial goal was to reduce the friction between the software developers and the compliance officers that temporarily join the agile team. The service enables the technical staff to capture information about the new or changed software artifacts and their structure at the time when the changes happen. The software created by 3rd parties is handled according to SOUP rules. The process is integrating into existing workflows and the tools used by the development team and serves as basic enablers for conducting risk management. To be able to perform these functions, the service provides features that fall in both Class A (clauses 4.3-c, 5.8.4, and 8.1.x) and Class B (clauses 5.3.1, 5.4.1, and 7.1.3). Manufacturers targeting Class A products may perceive this behaviour as excessive, because it involves activities related to software architectural design that goes beyond the minimum requirements in that class. However, from May 2020 when Medical Device Regulation [9] comes into force, the baseline for medical devices containing software is Class B, which makes our decision justified.

As mentioned in the previous subsections, other IEC 62304 clauses related to issue tracking, testing, and releasing are covered by other activities performed by the team as part of their DevOps pipelines and ceremonies. We decided that it would be beneficial to adopt and integrate those into the compliance activities rather than develop new tools.

VII. Conclusions

Bridging the medical device software development and agile practices is a long term endeavor that should be accomplished,

[13]https://www.atlassian.com/software/jira
[14]https://jenkins.io
[15]https://travis-ci.com
[16]https://circleci.com



(a) Clauses required by class A (green), B (orange), and C (red)

(b) Clauses covered by the service (green)

Figure 7. IEC 62304 clauses clustering

in truly agile spirit, in a sequence of small increments. The service implementation presented in this paper proves that properly designed tooling, although limited in scope, can bring together agile software development and compliance practices. The seamless integration into development team tools reduces friction, enabling the developers to focus on software development activities. The use of familiar DevOps patterns gives the developers the confidence that compliance problems are detected and handled by the responsible team members fast. Similarly, compliance officers are confident that they have up to date information about the software implementation in their area of interest. The high level of automation and transparency builds trust within the team. The tooling assists the team members to perform to compliance activities only when needed, making them efficient while maintaining high velocity. As a result, compliance becomes an organization's shared goal instead of an impediment.

REFERENCES

[1] Laukkarinen, T., Kuusinen, K., and Mikkonen, T. DevOps in regulated software development: case medical devices. In Proceedings of the 39th International Conference on Software Engineering: New Ideas and Emerging Results Track, pp. 15-18. IEEE Press, 2017.

[2] Anonymous, A. Anonymized title. IEEE Anonymized Workshop. IEEE, 2018.

[3] Cockburn, A. *Agile Software Development* Addison-Wesley, Boston, 2002.

[4] Eloranta, V.-P., Koskimies, K. and Mikkonen, T. Exploring ScrumBut – An empirical study of Scrum anti-patterns. Information and Software Technology 74, pages 194-203, Elsevier, 2016.

[5] Schwaber, K., and Beedle, M. *Agile software development with Scrum*. Vol. 1. Upper Saddle River: Prentice Hall, 2002.

[6] Debois, P. DevOps: A software revolution in the making. Journal of Information Technology Management 24, pages 3-39, no. 8, 2011.

[7] Fagerholm, F., Guinea, A.S., Mäenpää, H., and Münch., J. Building blocks for continuous experimentation. In Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, pp. 26-35. ACM, 2014.

[8] Official Journal of the European Union, Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation), Vol 119, 2016-05-04

[9] Official Journal of the European Union, Regulation (EU) 2017/745 of the European Parliament and of the Council of 5 April 2017 on medical devices, amending Directive 2001/83/EC, Regulation (EC) No 178/2002 and Regulation (EC) No 1223/2009 and repealing Council Directives 90/385/EEC and 93/42/EEC, Vol 117, 2017-05-05

[10] Centers for Medicare & Medicaid Services (1996), The Health Insurance Portability and Accountability Act of 1996 (HIPAA), Online http://www.cms.hhs.gov/hipaa/. Referred 15.7.2018.

[11] International Electrotechnical Commission, Medical device software - Software life cycle processes, IEC 62304, 2015-06

[12] International Standards Organization, Medical devices - Quality management systems - Requirements for regulatory purposes, ISO 13485, 2016-03-01

[13] International Standards Organization, Medical devices - Application of risk management to medical devices, ISO 14971, 2007-10-01

[14] Simon Brown, The C4 Model for Software Architecture, Online https://www.infoq.com/articles/C4-architecture-model. Referred 23.04.2019

[15] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In Proceedings of the 36th International Conference on Software Engineering (ICSE 2014). ACM, New York, NY, USA, 345-355. DOI: https://doi.org/10.1145/2568225.2568260